File Authorization with SMS and Fingerprint

December 7, 2016

Andrew Chen, Bryan Major, David Kuo (Team 9)

Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

andrewchenubc@gmail.com, bmajor@live.ca, davidkuo94@gmail.com

Abstract - This paper addresses the problem that arises with a typical file sharing service. It gives insight on an alternative method of authentication which uses a combination of fingerprint authentication and SMS messaging. An adversary model was created to show possible methods of attack. Security was evaluated based on this model, ensuring our design has sufficient countermeasures to defend against possible attacks outlined in the adversary model. Control groups were used to test the overall usability of our design over a period of 2 weeks and were asked to test different features. In our group, we split the implementation task into three parts: encryption / decryption (Bryan), server-sided code (Andrew) and client-sided code (David).

I. INTRODUCTION

Normally, secure file sharing is done by first establishing a shared session key, encrypting the file with this key, and transferring the encrypted file. However, this can get very cumbersome when the file needs to be sent to multiple people. This problem is solved with a file authorization system. A file authorization system allows for a user to securely share files to others. However, there are some drawbacks that we wish to address that are present in current file authorization systems. One of the biggest motivations for this project is the fact that all file authorization systems that we have researched have used a username and password pair for authentication. This is a major flaw as people often choose weak passwords or reuse passwords. Secondly, file authorizers are often impractical for large files. For most users, uploading a large file to a server is often restrictive due to constrained internet upload bandwidth. This is an even bigger problem when focusing on mobile applications. Mobile users are plagued with data restrictions and suffer from slower upload speeds compared to desktop computers. We wish to address this problem as well as the username and password issue in our design of a secure file authorizer.

There has been similar work done on file authorizers which emphasizes the usability of a fingerprint as an extra security layer with promising results [1]. However, the paper still utilizes a username and password pair for authentication, an issue we hope to solve with our design. Also, there is no mention of any security features such as protocols or encryption methods, which must be addressed if we are to make the system secure.

We begin by designing a system that uses SMS/push notifications (something you have) and fingerprinting (something you are) as an alternative to logging in using a username and password. This approach prevents weak passwords from being guessed. Optionally, we can increase usability by reading the text messages sent by this check. Fingerprinting itself is proven to be more usable than entering a username and password [1]. Instead of keeping files on the server itself, which can reduce usability as well as increase risk if the database were ever to be compromised, we instead keep a key pair of a hash of the encrypted file as well as the key that decrypts the encrypted file. Once the user is authenticated, they can choose to either add a hash file key pair or request a key from someone else. After receiving a key request, the key authorizer then authorizes themselves using the same authorization method as the requester. Once this is done, the key authorizer can choose to accept or deny the request by the key requester.

To evaluate our design, we have established a control group to test our product. Testing was done on a weekly basis. This was done to get insight on the usability of our design as we are trying to maximize usability without sacrificing security. This design is deemed successful if the results show that the extra security checks do not impede a user's willingness to use the program or if the user deems such methods as being highly usable and willing to sacrifice a bit of usability for the added security.

Results were positive in the fact that many preferred the combination of SMS and fingerprinting as a form of authorization. However, when compared to services such as Google drive, many preferred the Google service as they already had Google accounts.

From these results, we can conclude that although there are services like Google Drive that provide these services on the cloud, this service can provide a service that accomplishes cases that current cloud solutions do not (such as storing large files, lower risk of storing keys instead of files on cloud, and factoring out weak passwords).

To evaluate the security of our design, we first developed an adversary model. The model highlights what sort of capabilities an attacker would have when attempting to expose vulnerabilities. Using the adversary model, we looked at each section of our design, and performed an analysis on whether the attacker can compromise the system through that specific

channel using the capabilities provided by the adversary model. The information we want to protect against is the access of the file hash with the decryption key that is stored in the system. If the attacker is unable to get that information, we deem the system to be secure.

When designing this system, we made sure to follow the principles of secure design. Defense in depth is followed by using both a SMS check as well as the fingerprint reader to authorize the user. This adds an extra layer of security if the other is somehow compromised. Complete mediation is followed by ensuring that the user must be authenticated every time he / she adds or requests for a key pair. Psychological acceptability is used when considering if fingerprinting / SMS is more usable than the traditional username and password combination. Open design is demonstrated by this document.

Our design consists of 3 main sections, the mobile application, the server and the actual encryption and decryption of the file. Our team consists of 3 people, and thus each of us oversaw a specific part of the design. A table of the work is given below:

Name	Design Section	Responsibilities
Andrew	Server	• Storing user information in database • Issue and handle SMS challenge / response • Exposing HTTPS endpoints • Sending push notifications to mobile devices
Bryan	Encryption / Decryption	• Key generation • Encryption/Decryption
David	Client (Mobile Application)	• Handling user interaction with mobile application • Receive and process push notification and SMS messages • Send/Handle HTTPS requests/responses to/from server

II. Related Work

There has been a paper written by a UBC student which focuses on the usability of a fingerprint file authorizer [1]. Our designs are similar in the fact that both our systems use a fingerprint sensor as an authentication factor. Also, we are both designing a way to authorize file access to other people.

There are major differences in our design from the paper mentioned above. One of the major differences is storage of keys inside the server instead of the file itself as opposed to storing the file directly onto the system. Another difference arises because we built our design around the fact that we want an authentication method that does not use a username and password, something that is insecure in practice due to people oftentimes choosing weak or repeated passwords. Using the fingerprint scanner was just a result of this goal. Not only that, we have a second authentication factor through SMS messages. Lastly, the paper does not do a report on the actual security of such a file system, only the usability of fingerprinting. We are developing a secure method and thus all channels of entry are analyzed based on our adversary model.

There are not many forms of file authorizers available right now, such as Google Drive. However, Google Drive requires the use of a Google account. Not only that, it requires the user to specify other users to share files (in our design this may not be the case). Finally, once an account is given permission, the authorizer does not know when the users access the file, there is no history being kept.

III. Adversary Model

Before we analyzed the security of our design, we first constructed an adversary model that established the objectives and the capabilities of a possible attacker.

The objective for a possible attacker is to decrypt and access encrypted files without the permission of the file authorizer. The adversary aims to retrieve the keys that were used to encrypt the encrypted files and use the keys to gain unauthorized access to the files.

We decided that the adversary will have the following initial capabilities:

1) *Access to the mobile device:* The attacker will be able to gain physical access to a user's mobile device.
2) *Access to the encrypted file:* The attacker will be able to obtain a copy of the encrypted file. The encrypted file contains information such as the user id of the file authorizer as well as the hash of the file.

During the attack, the attacker will have the following capabilities:

1) *Man-in-the-Middle:* The attacker will be able to intercept, replay, or requests from mobile application to server
2) *Eavesdropping communications:* The attacker will be able to view the text messages and push notifications on the user's mobile device
3) *Alternate communication channel to the server:* The attacker will be able to craft and send HTTPS requests to the server outside of the mobile application.
4) *Swapping the SIM card:* The attacker will be able to swap the SIM card inside the user's device to change the registered phone number on the device.

IV. System Design

Our system revolves around the aspect of a password-free authentication system where the server stores a pair containing a file hash of an encrypted file and the corresponding key.

Each user has a list of "key pairs" that they have registered. This system of storing the file hash and key is highly efficient as the actual sizes of the file hash and the key are not very large. This would solve the issue of bandwidth restrictions, allowing for users with slow connections to use this application. Also, this increases usability for the user since the wait time to upload the file is decreased significantly. Not only is it efficient for storage, but it allows for the users to be more flexible in the choosing a file sharing medium (such as a physical medium (USB), or a fast sharing method (P2P sharing)).

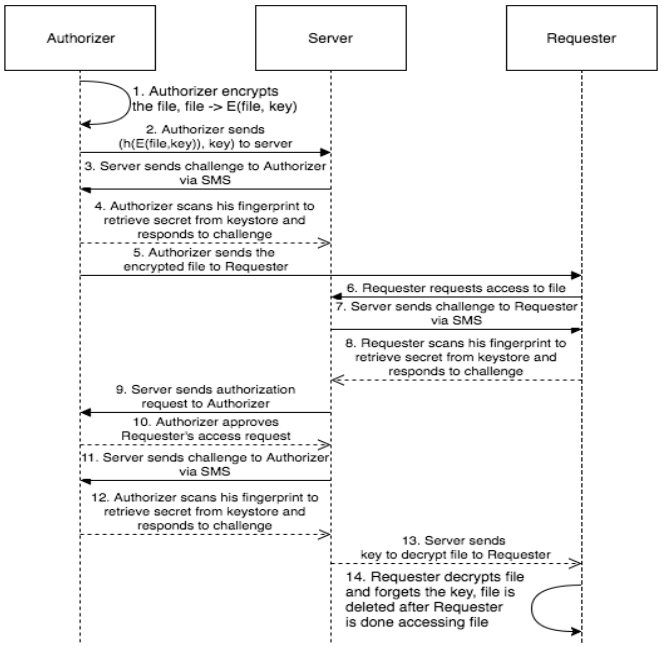

Fig. 1. File encryption and decryption flow diagram

Instead of a password, we use two factors of authentication, SMS messages and fingerprint checks. The server sends an SMS challenge to the user. Only the user with the correct phone number can get the SMS challenge. Next, the user must authenticate themselves using the mobile device's fingerprint scanner to unlock the device's unique fingerprint secret which was generated when the application was first started. Next, the application sends the fingerprint secret along with the SMS challenge to the server for authentication. The flowchart of the whole key authorization process can be seen in Figure 1.

To encrypt, a file is wrapped with its original file name, and a SHA-256 hash of the file with the original file name as seen in figure 2. A 128-bit AES key and 128-bit initialization vector (IV) are generated using the cryptographically strong random number generator. The key and IV are then used to encrypt the wrapped file using CBC mode with PKCS5 padding. The IV is then prepended to the encrypted file. Finally, the User ID is prepended to the file and can be saved with any file name. To decrypt a file, an access request is made to the user that corresponds to the user ID found in the encrypted file. Once the request is granted, the key is received from the server and the key is used to decrypt the file and the internal hash is checked. If the hash matches, the file is saved under the original filename.

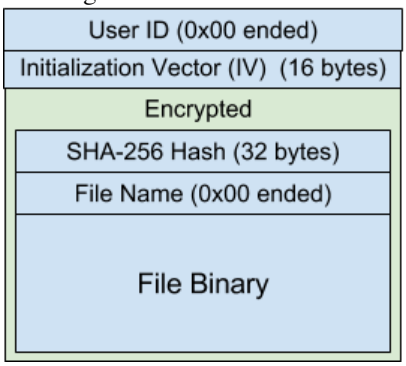

Fig. 2. File Wrapper and Encryption Format.

The following security principles important when considering the overall design of this system.

1) *Open Design:* We have followed Kerckhoff's Principle in that we want as much exposure to be able to fix any possible vulnerabilities that the design may have.
2) *Complete Mediation:* Complete mediation is important for our project since we do not want our users to read / write files that they should not have access to. The consequences might be the leaking of key pairs. The way we achieve complete mediation is by enforcing that the user authenticates itself whenever it does an action.
3) *Psychological Acceptability:* One of the most important aspects of the design is to be highly usable. If the application is not usable enough (i.e. it is too confusing or too complex), the user will use other methods of file authorization or even worse, no file authorization at all. Fingerprinting has been shown to be usable and so is SMS if we can implement reading of text messages.
4) *Defense in Depth:* Defense in depth is demonstrated by the two ways the user must identify themself. If the SMS is intercepted, we can still rely on fingerprint check and vice versa.

V. System Prototype

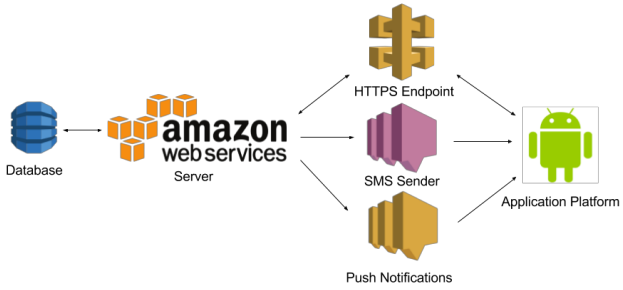

Fig. 3. High Level Design of the System Prototype

A. Cloud Server

We set up a cloud server that is hosted with Amazon Web Services (AWS) [2]. The cloud server is responsible for receiving requests from the applications, managing user accounts and authenticate the user for each user action. We chose amazon because it is an inclusive platform that contains all the features we need.

We created HTTPS API endpoints to communicate with the server using AWS API Gateway. The application can only communicate with endpoints using HTTP over SSL by default. We chose this because to not have to worry about generating our own certificate as well as being highly integrated with the rest of the AWS services. AWS API Gateway provided HTTPS endpoints to allow us to securely communicate with the server without having to worry about establishing an authentication protocol between the application and the server.

The server communicates with the client using push notifications and SMS messages. We used AWS Simple Notification Service (SNS) to drive the communication from the server to the client. AWS SNS integrates with Google's push notification service, Firebase Cloud Messaging to send push notifications to the user [3].

AWS Lambda is a server backend that delegates the other AWS services (such as storage of keys, sending notifications and handling endpoints). It features auto load balancing and metrics, so that a developer can focus on the code rather than miscellaneous problems.

For simplicity, we chose AWS DynamoDB, a fully managed NoSQL database service as our main storage method. This database is used to store keys as well as user data. With NoSQL, we do not need to worry about what sort of data is being passed in, it is all being handled by the database itself.

B. Android Application

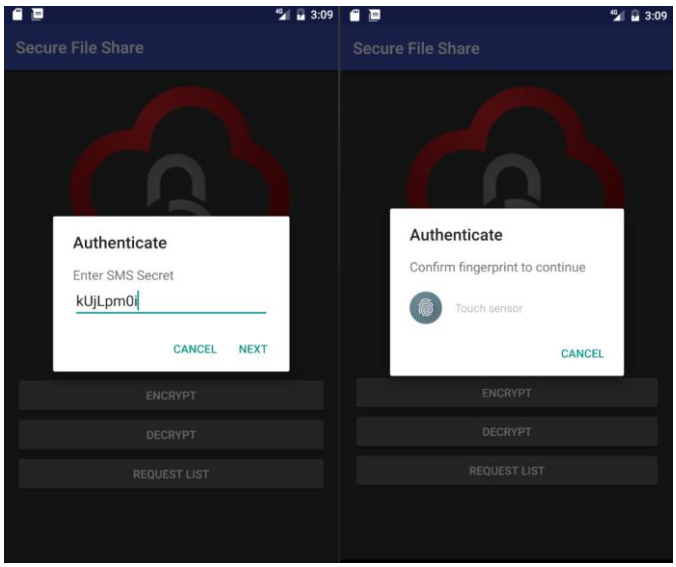

Fig. 4. Authenticating using SMS and fingerprint.

The Android application is responsible for encrypting and decrypting the file, receiving messages from the server through push notifications and SMS, sending requests to the server through HTTP requests, and authorizing user actions using the fingerprint scanner.

The Android application communicates with the server by sending HTTPS requests to exposed endpoints created with AWS API Gateway [2]. The application sends requests to the endpoints whenever it needs to communicate with the server.

The server sends notifications to the user through push notifications using Firebase Cloud Messaging (FCM) whenever the user receives a key or a key request [3]. When the Android application is first installed and launched, a firebase application token is generated. Additionally, a secret is generated and encrypted with a generated key that is stored in Android's Keystore System [4]. The secret is encrypted using "AES/ECB/NoPadding" mode. Since generated secret is 16 bytes in size, which is equivalent to 1 block in AES, no padding and no block chaining is required. Next, an account is created using the device's phone number, unique device identifier, the application token, and the unencrypted secret. The server sends push notifications to the application using the unique application token. The server uses SMS messages to send challenge codes to the application to authenticate the device for each action. The fingerprint scanner is used to authenticate the user and retrieve the generated secret before the application sends the device authentication code response to the server.

We are using Android's native file selector to handle the selecting of files to encrypt or decrypt [5]. To generate the key and IV we use the class "SecureRandom". Files are encrypted by the "Cipher" class in "AES/CBC/PKCS5Padding" mode.

The prototype creates several folders on the device while encrypting and decrypting files. A directory named "SecureFileShare", is created in "ExternalStorageDirectory", and all other directories are placed inside. Within "SecureFileShare" directory the app creates an "Encrypted" directory, "ToDecrypt" directory, and a "Decrypted" directory. The "Encrypted" and "Decrypted" directories are self-explanatory while the "ToDecrypt" directory stores encrypted files for which the user has made key requests. In the "ToDecrypt" directory the encrypted files are renamed to their SHA-256 hash so the file can be quickly found when a key is granted.

We decided to use the Firebase Cloud Messaging services because of the convenience in setting up the communication channels between the application and the server. Firebase Cloud Messaging simplifies sending push notifications to the application. Both the token generation and secure communication channels are handled by Google. We also decided to use the fingerprint scanner to authorize requests. This is due to the improvements in usability that the fingerprint scanner provided [1]. The fingerprint scanner allows the user to authenticate themselves without having to remember a long and secure password.

VI. Evaluation Methodology

A. Evaluation Methodology

We evaluated our project on two aspects: usability and security.

We evaluated the usability of our design by distributing the prototype application we have developed to our friends and family members to test. In total, we had a sample size of 15 people. We first asked each user to encrypt a file of their choice and to send the file to another user that is testing our application. The user that received the encrypted file then requested access from the authorizer. The second test involved providing each user with an already encrypted file to decrypt. Each user tested the decryption mechanism of our application by requesting access from the authorizer. The test period lasted for one week and we encouraged the testers to use the application at least once per day. At the end of each day, we followed up with each tester with a short questionnaire that aimed to assess the convenience of the password-less system. Following the end of the test period, we distributed a more detailed questionnaire to the testers to gather feedback on the overall experience and usability of our design. After, we reiterate on the design and make improvements if necessary.

We evaluated the security of our design based on the capabilities of a possible attacker outlined in our adversary model.

1) *Access to the mobile device:* Fingerprint scans are required for all actions taken on the application so the attacker cannot impersonate the user.
2) *Access to the encrypted file:* Given the encrypted file an attacker could request access to the file; however, since the phone number of the person requesting a file is shown to the authorizer, the authorizer can identify bad requests.

The attacker could also try to decrypt the file however since AES-128 is secure the hacker would have to resort to a brute force attack which is infeasible.

3) *Man-in-the-Middle:* Communication between the server and the application is secured using TLS with certificates provided by Google and Amazon. Users must pass an authentication check first before doing any user actions. For each authentication process, the user must also scan their fingerprint to unlock the key in the Android Keystore and use the key to decrypt the encrypted copy of the generated secret. The challenge responses are then sent containing a corresponding request number for the original request, the SMS challenge code, and the generated secret. This process ensures the attacker cannot receive data with forged HTTP requests.
4) *Eavesdropping on communications:* Since all communication between the server and application is encrypted using TLS, the attacker cannot gain meaningful information from eavesdropping on the communication between the application and the server.
5) *Alternative communication channel to the server:* Like Man-in-the-Middle, the attacker cannot retrieve any valuable data or prompt any actions on the server since authentication must be done using the physical device of the user and must bypass the fingerprint scanner of the device.
6) *Swapping the SIM card:* The attacker cannot authenticate properly to the server. If the attacker replaces the SIM card of the stolen device with their own, the SMS notification challenge will not pass since the SMS notification is sent to the phone number of the original SIM card. If the attacker uses the stolen SIM card with their own device, the generated secret will not match since the attacker's own device does not have the same secret as the stolen device.

B. Results of the Evaluation

When we did the first round of evaluations, we were surprised that people thought our app was unusable. Based on the rating system we gave them, we first asked them if they preferred SMS and fingerprinting to accounts and got the following results:

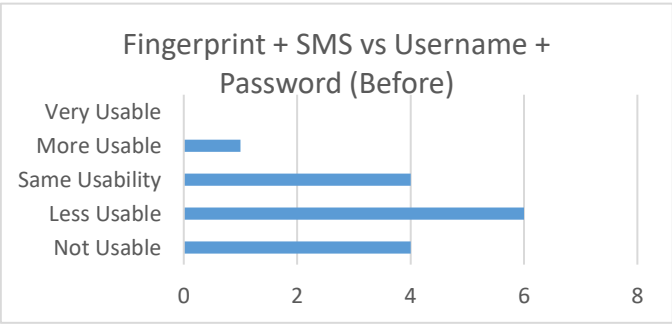

Fig. 5. Usability of SMS and Fingerprint (without Auto-SMS read).

After discussing what made the app unusable, we realized that typing in the SMS secrets in manually wasn't feasible since the user had to constantly swap between the SMS messaging app and our app. The result of this act was that a lot of time was wasted during authentication. The users have also expressed their frustration interpreting the SMS secret (which is an 8-character alphanumeric password). For the people who did not copy and paste the message, they often misinterpreted the lower-case letter "L" with the number '1" or the upper case "I". To address these user pain points, we implemented a way to read the SMS messages from the phone as it receives messages. Although this requires an extra permission check when you use the application for the first time, it allows for a more streamlined process as the SMS check would not need any user input to pass. After redoing the evaluations, we got the following results:

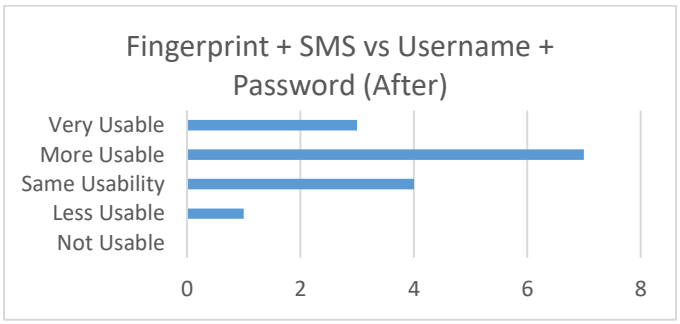

Fig. 6. Usability of SMS and Fingerprint (with Auto-SMS read).

Next, we asked them to compare this service to the file authorization service provided by Google. We got the following results:

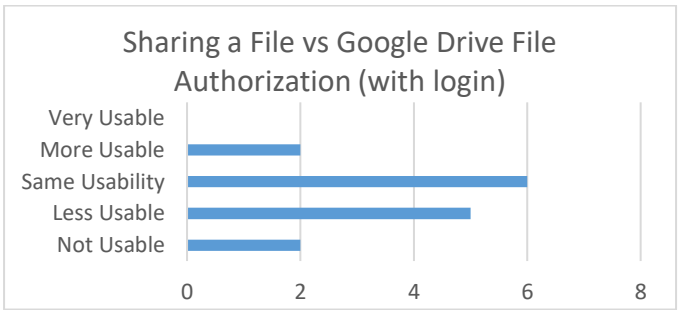

Fig. 7 Usability of our design vs Google File Authorization.

C. Discussion of Evaluation Results

After making the changes to the design we found a much more positive result in regards to usability. The combination of fingerprint and SMS was found to be superior to the usage of accounts to log in. This was largely because automatic SMS reading was added. This result agrees with the paper focusing on fingerprinting as an authentication tool [1]. We both conclude that fingerprinting is more usable than typing in a password every time. This makes sense since our SMS check does not require any extra work to the user.

This design was deemed to be less usable than simply clicking a link to Google Drive. This is because in Google Drive, we do not have to wait for the authorizer to approve before we can access the file. Although this is a property of all file authorizers, the users preferred quick access over security of the file. Google Drive also can make the user wait for the authorizer to approve first before allowing them to download. Most people preferred this service since they already have a Google account memorized. However, the people who did not have a Google account either thought our

application had the same or slightly better usability than that of Google Drive.

VII. Discussion

1) Advantages of our Design

Following our adversary model, a user has many advantages when using our design in comparison to other similar applications available today. The user can control the access of their file at an individual level. Google Drive offers this level of access control as well [6]. However, Google Drive's offering is tied directly to a user's email account. In our design, the access is tied to a user's phone number and biometrics. The user can only send access requests to the file owner by authenticating themselves using SMS and their fingerprint.

An adversary may steal the file requestor's device or spoof the phone number of the file requestor, the fingerprint authentication process must be done before the request is sent to the file owner. This way, the file owner can be sure that any requests made to access the file were done by the owner of the mobile device that made the request. These features allow the file owner to not worry about their encrypted files being compromised if they ever lose their mobile device.

Additionally, our design does not restrict the user in terms of file storage medium. Once our design encrypts the file, the owner can freely choose which medium to use when sharing the encrypted file. This can be physical media, such as a USB stick, or online services such as email. The file can be shared publicly without worry since the file is encrypted. With Google Drive, the user must first upload their file onto their cloud storage, and then provide access to the file via a shareable link. The user can then choose to enforce access control rules when creating the shareable link. In this case, the user is forced to use Google Drive as a hosting service.

Finally, our design requires the user to authenticate their actions using a combination of an SMS challenge code and an encrypted, generated secret that is decrypted using fingerprint authentication. This means the user does not have to worry about managing and memorizing another user account.

2) Disadvantages of our Design

Our design requires the user to use the mobile application to encrypt and decrypt the files. This limitation is shared with the system developed by G. Lam, but not with Google Drive [1]. Additionally, the user's mobile device must contain a fingerprint reader to utilize this application. However, this limitation will be less of a problem as fingerprint sensors become more prevalent in smartphones.

There are also disadvantages associated with an account-free design. Due to the lack of accounts, a user must authenticate themselves for each action that involves private information. Therefore, user must authenticate themselves multiple times for each session. However, a method that we have employed to combat the issue of SMS challenges is to use automatic SMS reading. From our evaluation results, we can see that users find SMS challenges much more usable once automatic SMS reading is implemented into the design.

Another option to improve the usability of the design is to compromise complete mediation. If we were to lessen the mediation of the design the following changes could be done:

1) *Only verify phone number with an SMS on account creation.*
2) *Implement a session-based secret that is established once the user has authenticated for the first time*

However, the design change does not protect against a situation where the user's phone is compromised with the application opened.

VIII. Conclusion

Our goal was to design a secure method to do file sharing without the use of accounts. Our design focuses on maximizing usability while still being secure to attacks. Through our results, we see that although it might not be as usable to others as a simple Google Drive link, some people are willing to sacrifice a bit of usability in exchange for security and non-repudiation. As a bonus, our design handles cases that are not supported by cloud services (such as large files and freedom of file sharing medium). This work is important in the development of a method that would one day phase out the use of passwords, because passwords are highly unreliable.

The current design prototype can be improved with the following extensions:

1) *Cross platform*: Currently, the prototype is only supported on Android. The prototype can be extended to support the iOS platform
2) *Session-based tokens*: Implementing session based tokens will immensely improve usability as the user will not have to authenticate for each action
3) *One-time use files:* Currently, the prototype does not automatically delete the file once the user has finished using the file. This will improve the security as access can be controlled up to usage attempts
4) *File Names*: In addition to the file hash, we can keep the file name of the file so that the user can easily identify the file being requested.

References

[1] G D Lam, "Evaluating the Usability of an Apple Touch ID-Based Access Control System" University of British Columbia, April 2015.

[2] "Amazon Web Services (AWS) – Cloud Computing Services," *Amazon Web Services, Inc*. [Online]. Available: https://aws.amazon.com/. [Accessed: 10-Nov-2016].

[3] "Firebase | App success made simple," *Firebase*. [Online]. Available: https://firebase.google.com/. [Accessed: 10-Nov-2016].

[4] A. System, "Android Keystore System | Android Developers", Developer.android.com, 2016. [Online]. Available: https://developer.android.com/training/articles/keystore.html. [Accessed: 02- Dec- 2016].

[5] O. Framework, "Open Files using Storage Access Framework | Android Developers", Developer.android.com, 2016. [Online]. Available: https://developer.android.com/guide/topics/providers/document-provider.html. [Accessed: 02- Dec- 2016].

[6] "Latest Google Drive App Update Lets You Request Access To Files You Don't Have Access To", Android Police, 2016. [Online]. Available: http://www.androidpolice.com/2015/10/09/latest-google-drive-app-update-lets-you-request-access-to-files-you-dont-have-access-to/. [Accessed: 02- Dec- 2016].

[7] "ChenAndrew/CPEN442-Security-Project", GitHub, 2016. [Online]. Available: https://github.com/ChenAndrew/CPEN442-Security-Project. [Accessed: 07- Dec- 2016].

[8] "davidk894/cpen442_project", GitHub, 2016. [Online]. Available: https://github.com/davidk894/cpen442_project. [Accessed: 07- Dec- 2016].